LRPC: Lightweight Remote Procedure Call

by B.N. Bershad, T. Anderson, E. Lazowska, H. Levy

- What is it: communication facility designed and optimized for *cross-domain* communication within the same machine
- Reduces the cost of traditional RPC calls
- Lightweight
- Combines:
  - control transfer and communication model of capability systems
  - programming semantics and large-grain protection model of RPC
- Performance improvements for same machine communication and simple arguments

Preliminaries

Protection:

- two-level: separate O.S. from user program through hardware
- capabilities
  - fine-grained protection
    - each object exists in its own protection domain
    - but all live within a single name or address space
    - a process from one domain can act on an object of another domain by making a *protected* procedure call
  - large-grain protection
    - RPCs
    - popular because of secure communication across machines

Small-Kernel Operating Systems

- Borrow the large-scale protection and programming models used in distributed computing systems
- Use them to manage subsystems
- Idea: separate O.S. components can be placed in disjoined domains (or address spaces), then use messages for *inter-domain* communication
- Advantages
  - modular, non-monolithic
  - easy system design
  - debugging
  - decomposable
  - implementation
  - maintenance
  - failure isolation
  - etc

- Communication traffic is mostly cross-domain
- Localize processing and resources for performance
- Simple communication
- Treat local communication differently than remote communication
- Otherwise, you treat simple operations like complex ones...
- Anyway, if you cram too much within the same domain, size and complexity increase
- ⇒ facilitate cross-domain communication: LRPC
  - simple control transfer
  - simple data transfer
  - simple stubs
  - concurrency

Use and performance of RPC systems

- Small fraction of RPC calls are truly remote
- Mostly, cross-domain
- Measurements of cross-machine
  - V system (3 %)
  - Taos (DEC Firefly, 5.3%)
  - Unix and NSF, even less:
    RPC is only used to access remote file servers only...
- Measurements of parameter size and complexity of cross-domain
  - most of them to the *same* procedures
  - number of bytes: small! bytes copying was enough

Use and performance of RPC systems (cont.)

- Performance of cross-domain RPCs
- Poor!
- Uses the facilities of cross-machine ones!
- Example

```
procedure Null();
begin
return;
end;
```

- Cross domain: one procedure call, kernel trap and change of virtual address context, then a trap and context change in return
- Significant more if we use RPCs...

RPC Overheads

- stub overhead
- message buffer overhead
- access validation
- message transfer (enqueue, dequeue, at sender/receiver)
- scheduling
- context switch
- dispatch
- trade-offs: to reduce copying, message buffers can be globally shared (SRC RPCs) but weaker security...

Design and Implementation

- Execution model: protected procedure call
  - kernel validates caller, creates call linkages, dispatches client's thread directly to server domain...
  - argument stack is provided and the end results return through kernel back to the point of client's call...
- Programming semantics and large grain protection: RPC
  - servers execute in private protection domain
  - make a set of procedures available to other domains
  - binding: authorization

Binding

- High-level, similar to RPC
- Lower-level: different
  - clerk registers an interface with a name server
  - then, waits import requests from clients
  - client binds to a specific interface with an import call
  - waits until the kernel notifies the server's waiting clerk
  - binding is enabled with a procedure descriptor list (PDL) maintained by the exporter at each interface
  - PDL contains one procedure descriptor (PD) for each procedure
  - PD: entry address, number of permitted simultaneous calls, argument stack (*A-stack*)

  - for each PD the kernel allocates in the client and server domains a number of A-stacks equal to the number of simultaneously allowed calls
  - A-stack: mapped read-write and shared by both client and server domains
  - linkage record for each A-stack: where to return
  - if procedures in the same interface have stacks of the same size, can share A-stacks: save storage needs if interfaces have multiple procedures

Calling

- Stubs at both domains
- Client calls its stub, which manages the A-stack (crams all info needed there, then traps to the kernel)
- Verifications/linkage is done by the kernel
- Finds an execution stack *E-stack* in the server's domain
- Updates the thread's user stack to point to the E-stack
- Reloads the processor's virtual memory registers with those of the server domain
- Calls the server's stub at the address specified by the procedure descriptor

- Associate A-stack/E-stack – association remains till the calls returns
- Since A-stack is mapped into the server's domain, this means that the server procedure can directly access the parameters as though it had been called directly

Stub Generation

- Automatically generated
- Run-time stubs in assembly
- Purpose: blur boundaries between protocol layers and reduce cost of traversing them...

LRPC on Multiprocessors

- Higher call TPUT and lower call latency
- Minimize the use of shared data structures on the critical domain transfer path
- Shared memory systems
- Main idea: reduce LRPC latency by caching domain contexts on idle processors
- Kernel checks for idle processor in the server domain
- If one is found, the kernel "exchanges" the processors of idling and calling threads, placing the calling thread on a processor where the context is already loaded

- The called server procedure can then execute on that processor without a context switch
- The idling thread is still idle, but on the context of the client domain
- Advantage: no context switch

Argument copying

- Standard RPC: 4 times
- LRPC: just once
  - Client-stub to to the shared A-stack
  - Pair-wise allocation of A-stack is a plus
  - Ensures protection
  - Avoid parameter copy when unnecessary (ex. write)
  - Comparisons

Performance

- LRPC/MP: uses idle processor optimization
- Regular context switch on a single processor

Summary

- Exploits simple control transfer
- Exploits simple data transfer
- Simple stubs
- Multiprocessors
- Avoids needless scheduling
- Avoids unnecessary access validation
- Avoids redundant copying and lock contention
- Alternative for small kernel systems...